

Ringkasan Materi Pemrograman Berorientasi Objek (PBO)

Pengantar, Konsep Dasar, hingga Polimorfisme

Pahrul Irfan, M.Kom

Agenda Pembelajaran

01

Pengantar Pemrograman & PBO

Paradigma Pemrograman, Definisi dan Keunggulan PBO, Konsep Dasar: Class, Objek, Atribut, Method

02

Dasar Pemrograman Java

Pengenalan Java & Struktur Program, Tipe Data, Variabel, dan Operator, Struktur Kontrol: Percabangan & Perulangan, Konsep Array

03

Konsep Inti PBO

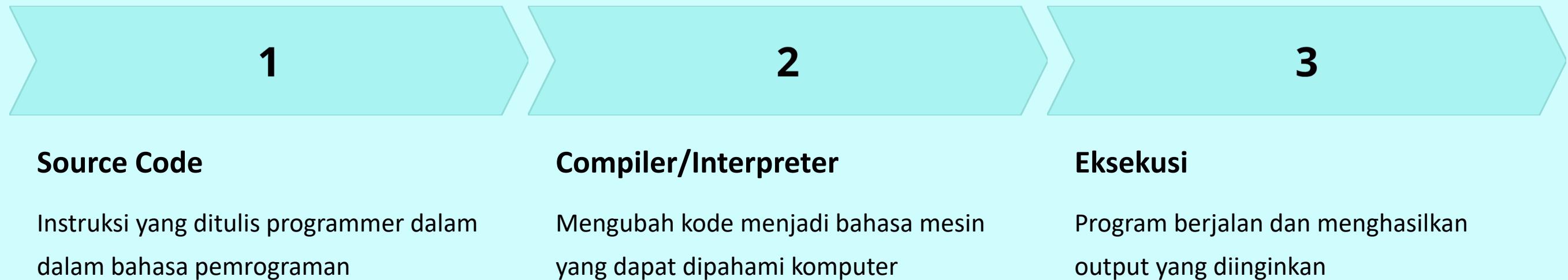
Constructor, Encapsulation (Pengkapsulan), (Pengkapsulan), Inheritance (Pewarisan), (Pewarisan), Polymorphism (Polimorfisme) (Polimorfisme)

BAGIAN 1

Pengantar Pemrograman & PBO

Mengapa Belajar Pemrograman?

Komputer hanya memahami bahasa mesin (angka 0 dan 1). Manusia sulit berkomunikasi langsung dengan bahasa ini. **Bahasa Pemrograman** **Bahasa Pemrograman** bertindak sebagai "penerjemah" antara bahasa manusia dan bahasa mesin.



Java menggunakan keduanya: javac (compiler) mengubah .java menjadi bytecode (.class), lalu java (interpreter/JVM) menjalankan bytecode menjalankan bytecode tersebut.

Paradigma Pemrograman

Setiap programmer memiliki cara berbeda dalam menulis instruksi. Beberapa pendekatan utama adalah:

Prosedural/Imperatif

Fokus pada urutan instruksi dan manipulasi data secara bertahap

Fungsional

Program dilihat sebagai entitas input-output dan dibagi menjadi fungsi-fungsi kecil

Berorientasi Objek (PBO)

Program dilihat sebagai kumpulan objek yang saling terkait dan berinteraksi

Apa itu PBO? Dan Mengapa PBO?

Definisi: PBO adalah sebuah pendekatan pengembangan perangkat lunak yang seluruh strukturnya berbasis pada konsep **objek**. Data dan aksi (fungsi) yang bekerja pada data tersebut disatukan dalam sebuah unit bernama objek.



Pemeliharaan Efisien

Struktur yang terorganisir memudahkan pemeliharaan



Pengembangan Cepat

Komponen (objek) dapat digunakan kembali (reusability)



Mendukung Kerja Tim

Pembagian tugas menjadi jelas antar modul/objek



Model Dunia Nyata

Mudah menerjemahkan model bisnis ke dalam model pemrograman

Konsep Fundamental: Class dan Objek

Class

Sebuah *blueprint*, cetakan, atau rancangan yang mendefinisikan struktur dan tingkah laku dari sebuah objek. Merupakan deskripsi statis dari sesuatu.

Objek

Adalah *instance* atau perwujudan nyata dari sebuah class. Setiap objek bersifat independen meskipun berasal dari class yang sama.

- ❑ **Analogi:** Class adalah **cetakan kue**, sedangkan objek adalah **kue** yang dihasilkan. Warna kue bisa berbeda-beda meskipun cetakannya sama.

Anatomi Class: Atribut dan Method

Setiap objek yang dibuat dari sebuah class memiliki:

Atribut (State/Properti/Variabel)

Data yang dimiliki oleh objek, yang mendeskripsikan karakteristiknya.

- Contoh pada objek Mobil: warna, tahunProduksi
- Contoh pada objek Mahasiswa: nim, nama

Method (Behavior/Aksi/Fungsi)

Aksi yang dapat dilakukan oleh objek.

- Contoh pada objek Mobil: menghidupkanMesin(), menjalankanMobil()
- Contoh pada objek Mahasiswa: mengambilMataKuliah()

BAGIAN 2

Dasar Pemrograman Java

Pengenalan Bahasa Java

Karakteristik: Java adalah bahasa yang kuat, sederhana, berorientasi objek murni, dan aman.

Platform Independent: Dikenal dengan slogan "*Write once, run anywhere*". Kode Java (.java) (.java) dikompilasi menjadi *bytecode* (.class) yang dapat dijalankan di platform mana pun yang memiliki yang memiliki **Java Virtual Machine (JVM)**.

Variabel dan Tipe Data

Variabel: Lokasi di memori komputer untuk menyimpan nilai yang dapat berubah. Variabel harus dideklarasikan dengan tipe data tertentu.

```
tipeData namaVariabel; // Deklarasi  
int jumlah = 10; // Inisialisasi
```

Tipe Data di Java dibagi menjadi dua kategori besar:

Tipe Data Primitif

Tipe data dasar yang tertanam di Java, menggunakan huruf kecil

Tipe Data Referensi (Objek)

Tipe data berupa class, baik dari library Java maupun buatan sendiri, diawali huruf kapital.
Contoh: String

Tipe Data Primitif

Tipe data ini menyimpan nilai sederhana dan diproses lebih cepat.

#

Bilangan Bulat

- byte: 8-bit (-128 hingga 127)
- short: 16-bit (-32.768 hingga 32.767)
- int: 32-bit (sekitar -2 miliar hingga 2 miliar), paling umum
- long: 64-bit (untuk angka yang sangat besar)

¶

Bilangan Pecahan

- float: Presisi 32-bit
- double: Presisi 64-bit, lebih umum dan akurat

@@@

Karakter

char: 16-bit, untuk satu karakter Unicode, ditandai dengan kutip tunggal ('A')

@@

Logika

boolean: Hanya menyimpan nilai true atau false

Tipe Data Primitif

Tipe Data	Ukuran (bits)	Rentang Nilai	Contoh
byte	8 bit	-128 sampai 127	byte a = 100;
short	16 bit	-32,768 sampai 32,767	short b = 1000;
int	32 bit	-2,147,483,648 sampai 2,147,483,647	int c = 50000;
long	64 bit	-9,223,372,036,854,775,808 sampai 9,223,372,036,854,775,807	long d = 100000L;
float	32 bit	6-7 digit desimal	float e = 5.75f;
double	64 bit	15 digit desimal	double f = 19.99;
char	16 bit (Unicode)	Karakter tunggal	char g = 'A';
boolean	1 bit (logis)	true atau false	boolean h = true;

Tipe Data Reference

- Reference type **menyimpan alamat (referensi)** ke objek di memori (heap).
- Tidak menyimpan nilai langsung (beda dengan primitive).
- Semua class di Java termasuk tipe data reference.

Tipe Data Refrence

String → untuk teks (misalnya: "Hello Java")

Array → sekumpulan nilai dengan tipe yang sama (misalnya: int[] angka = {1,2,3};)

Class → cetakan untuk membuat objek (misalnya: class Mahasiswa {})

Object → instance dari class

Wrapper classes → versi object dari primitive (misalnya: Integer, Double, Character)

Tipe Data Refrence (String)

```
String nama = "Andi"; // reference ke object String  
// method milik class String  
System.out.println(nama.toUpperCase());
```

Tipe Data Refrence (Array)



Main.java

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int[] angka = {1, 2, 3, 4}; // reference ke array object  
4         System.out.println("Panjang array: " + angka.length);  
5     }  
6 }  
7
```

Tipe Data Reference (class)

```
class Mahasiswa {  
    String nama;  
    int umur;  
    // constructor  
    Mahasiswa(String n, int u) {  
        nama = n;  
        umur = u;  
    }  
    void info() {  
        System.out.println("Nama: " + nama + ", Umur: " + umur);  
    }  
}  
public class Main {  
    public static void main(String[] args) {  
        Mahasiswa mhs = new Mahasiswa("Budi", 20); // reference ke object Mahasiswa  
        mhs.info();  
    }  
}
```

Tipe Data Refrence (Array)

```
Integer x = 10;
```

```
System.out.println(x.toString());
```

Operator dalam Java

Aritmatika

+ (penjumlahan), - (pengurangan), *
* (perkalian), / (pembagian), %
(modulus/sisa bagi)

Penugasan (Assignment)

= untuk memberi nilai. Bisa digabung:
digabung: +=, -=, *=

Increment/Decrement

++ (tambah 1), -- (kurang 1). Bisa pre
pre (++x) atau post (x++)

Relasional (Pembanding)

== (sama dengan), != (tidak sama dengan), >, <, >=, <=.
<=. Menghasilkan nilai boolean

Logika

&& (AND), || (OR), ! (NOT). Untuk menggabungkan ekspresi
boolean

Struktur Kontrol: Percabangan (Seleksi)

Memungkinkan program memilih alur eksekusi berdasarkan kondisi.

if

Menjalankan blok kode jika kondisi true

```
if (nilai > 60) {  
    System.out.println("Lulus");  
}
```

if-else if-else

Untuk beberapa kondisi berurutan

```
if (nilai > 85) {  
    // A  
} else if (nilai > 70) {  
    // B  
} else {  
    // C  
}
```

if-else

Menyediakan alur alternatif jika kondisi false

```
if (angka % 2 == 0) {  
    // genap  
} else {  
    // ganjil  
}
```

switch

Alternatif untuk if-else if jika membandingkan satu membandingkan satu variabel dengan banyak nilai banyak nilai konstan

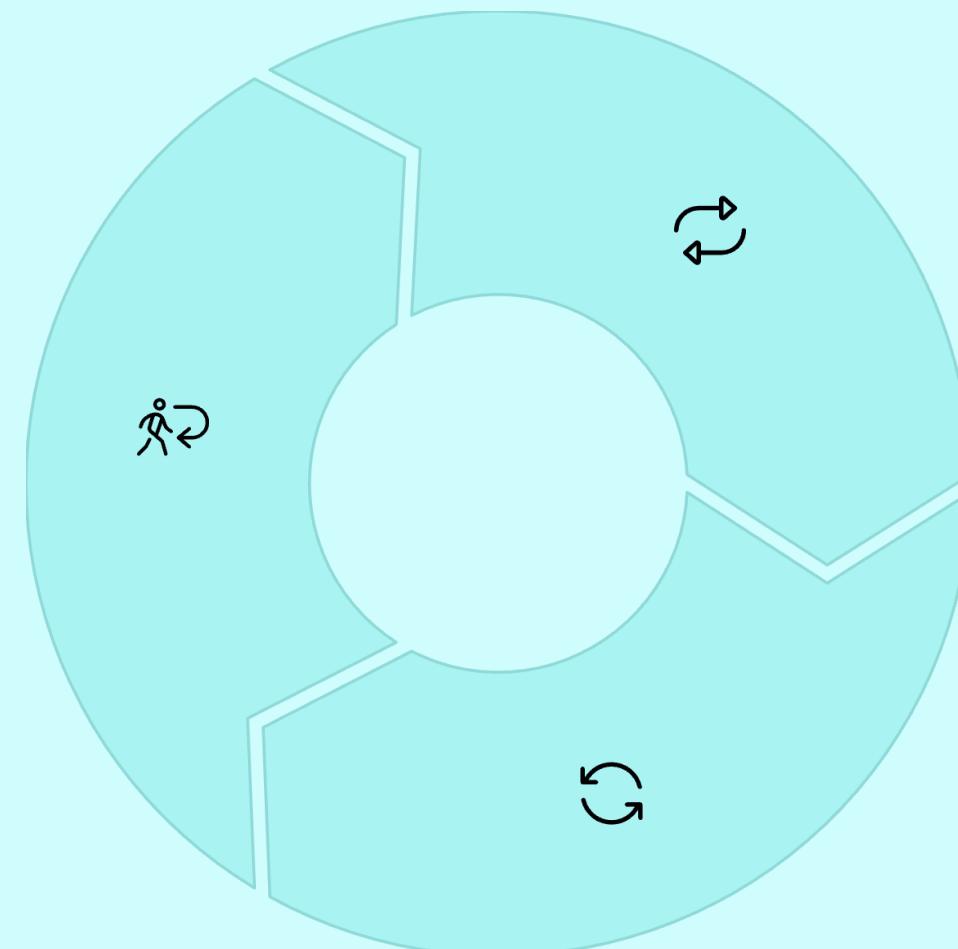
Struktur Kontrol: Perulangan (Looping)

Mengeksekusi blok kode secara berulang selama kondisi terpenuhi.

while

Mengecek kondisi **di awal**.
awal. Blok kode mungkin
tidak akan pernah dieksekusi.
dieksekusi.

```
int i = 0;  
while (i < 5) {  
    System.out.println(i);  
    i++;  
}
```



do-while

Mengecek kondisi **di akhir**.
akhir. Blok kode dijamin
dieksekusi **minimal satu kali**.
kali.

for

Paling umum digunakan ketika
ketika jumlah iterasi sudah
diketahui. Menggabungkan
Menggabungkan inisialisasi,
inisialisasi, kondisi, dan update
update dalam satu baris.

Pengantar Array

Konsep

Array adalah sebuah objek yang digunakan untuk menyimpan **kumpulan data sejenis** dalam jumlah elemen yang tetap.

Manfaat

Menghindari pembuatan banyak variabel untuk data serupa. Misalnya, menyimpan 100 nilai mahasiswa nilai mahasiswa cukup dengan 1 variabel array, bukan 100 variabel int.

Indeks

Setiap elemen di dalam array diakses melalui indeks, yang dimulai dari **0**. Jika array memiliki 10 elemen, memiliki 10 elemen, indeksnya dari 0 hingga 9.

Deklarasi dan Penggunaan Array

Deklarasi & Inisialisasi:

```
1 // Cara 1: Deklarasi, lalu alokasi memori
2 int[] nilaiMahasiswa; // Deklarasi
3 nilaiMahasiswa = new int[10]; // Alokasi 10 elemen
4
5 // Cara 2: Deklarasi dan alokasi dalam satu baris (umum)
6 double[] suhu = new double[7];
7
8 // Cara 3: Deklarasi dengan nilai awal
9 String[] hari = {"Senin", "Selasa", "Rabu"};
```

Deklarasi dan Penggunaan Array

1

Mengakses Elemen

Gunakan nama array diikuti indeks dalam kurung siku []

```
// Mengisi elemen pertama
```

```
suhu[0] = 36.5;
```

```
System.out.println(hari[1]);
```

```
// Mengambil "Selasa"
```

2

Panjang Array

Properti .length digunakan untuk mengetahui jumlah elemen

```
for (int i = 0; i < hari.length; i++) {
```

```
    System.out.println(hari[i]);
```

```
}
```

Array Multidimensi

Array juga dapat memiliki lebih dari satu dimensi, sering disebut "array dari array". Array 2D sering digunakan untuk merepresentasikan data dalam bentuk tabel atau matriks.

Deklarasi & Penggunaan:

```
1 // Array 2D untuk menyimpan matriks 3x3
2 int[][] matriks = new int[3][3];
3 matriks[0][0] = 1; // Mengisi elemen baris 0, kolom 0
4
5 // Deklarasi dengan nilai awal
6 String[][] nama = { {"Pak ", "Bu "}, {"Joko", "Susi"} };
7 System.out.println(nama[0][0] + nama[1][0]); // Output: "Pak Joko"
```

Untuk mengakses semua elemen, biasanya digunakan *nested loop* (perulangan bersarang).

Pengantar Array

	Array	ArrayList	Vector
Ukuran	Fixed (tidak bisa diubah setelah dibuat)	Dinamis (bisa bertambah/berkurang)	Dinamis (bisa bertambah/berkurang)
Tipe Data	Bisa primitif & object	Hanya object (gunakan wrapper utk primitif)	Hanya object (gunakan wrapper utk primitif)
Properti dasar	.length → jumlah elemen	.size() → jumlah elemen sekarang	.size() → jumlah elemen sekarang
Kapasitas	Tidak ada (selalu fixed)	.size() dan .ensureCapacity()	.size() dan .capacity() (kapasitas internal saat ini)
Tambah elemen	✗ Tidak ada .add()	<input checked="" type="checkbox"/> .add(obj)	<input checked="" type="checkbox"/> .add(obj)
Hapus elemen	✗ Tidak ada .remove()	<input checked="" type="checkbox"/> .remove(index)	<input checked="" type="checkbox"/> .remove(index)
Akses elemen	array[i]	.get(i)	.get(i)
Ubah elemen	array[i] = x	.set(i, x)	.set(i, x)

BAGIAN 3

Konsep Inti PBO

Constructor - "Pembangun" Objek

Definisi: Sebuah **method khusus** yang dipanggil secara otomatis saat sebuah objek baru dibuat (new).

Fungsi Utama: Melakukan **inisialisasi**, yaitu memberikan nilai awal pada properti objek.



Ciri-Ciri Constructor

- Nama constructor **sama persis** dengan nama kelas
- Tidak memiliki *return value*, bahkan tidak menggunakan void
- Bisa memiliki parameter (*parameterized constructor*) atau tanpa parameter (*default constructor*)

Overloading Constructor

Overloading: Kemampuan membuat beberapa method dengan **nama yang sama**, tetapi **jumlah atau tipe jumlah atau tipe parameternya berbeda**.

Prinsip ini juga berlaku untuk constructor, sehingga sebuah class bisa memiliki lebih dari satu constructor.

- **Tujuan:** Memberikan fleksibilitas saat membuat objek. Misalnya, membuat objek Mahasiswa hanya dengan NIM, atau dengan NIM dan nama sekaligus.

```
public class Mahasiswa {  
    // Constructor 1: tanpa parameter  
    public Mahasiswa() { ... }  
  
    // Constructor 2: dengan NIM saja  
    public Mahasiswa(String nim) { ... }  
  
    // Constructor 3: dengan NIM dan nama  
    public Mahasiswa(String nim, String nama) { ... }  
}
```

Encapsulation - Menyembunyikan Detail

Konsep: Cara untuk menyembunyikan detail implementasi dan melindungi data dari akses langsung yang tidak terkontrol.

Tujuan: Melindungi data, membuat kelas lebih mudah dikelola, dan mengurangi potensi *bug*.

01

Jadikan Atribut Private

Semua atribut/properti dibuat private untuk mencegah akses langsung

02

Sediakan Method Getter

Method public untuk mengambil nilai nilai (accessor)

03

Sediakan Method Setter

Method public untuk mengubah nilai nilai (mutator) dengan validasi data data

Access Modifiers

Access modifiers menentukan tingkat aksesibilitas dari kelas, properti, dan method.



private

Hanya dapat diakses **dari dalam kelas itu sendiri**. Tingkat akses paling ketat.



default

Hanya dapat diakses oleh kelas-kelas lain dalam **paket yang sama sama** (*package-private*).



protected

Dapat diakses oleh kelas dalam paket yang sama dan oleh subclass-nya (walaupun di paket berbeda).



public

Dapat diakses dari mana saja. Tingkat akses terluas.

Inheritance - Mewariskan Sifat

Konsep: Memungkinkan sebuah kelas baru (**subclass/child class**) untuk **mewarisi atribut dan method** dari kelas yang sudah ada (**superclass/parent class**).

Ini adalah implementasi dari relasi "*is-a*" (adalah sebuah). Contoh: Manager **adalah sebuah** Employee.



Menghindari Redundansi Kode

Fitur umum cukup ditulis sekali di superclass



Reusability

Menggunakan kembali kelas yang ada untuk membuat yang baru



Memudahkan Pemeliharaan

Perubahan di superclass otomatis berlaku untuk semua subclass

Implementasi Inheritance: extends dan super

Keyword extends

Digunakan untuk mendeklarasikan sebuah kelas sebagai turunan dari kelas lain.

```
public class Mobil extends Kendaraan { //  
Mobil mewarisi semua atribut dan method dari Kendaraan  
}
```

Keyword super

Digunakan untuk merujuk ke anggota (atribut, method, constructor) dari **superclass**.

- `super(parameter)`: Memanggil constructor *superclass*
- `super.namaMethod()`: Memanggil method dari *superclass*

Polymorphism - "Banyak Bentuk"

Definisi: Secara harfiah berarti "banyak bentuk". Ini adalah kemampuan suatu variabel dari **supertype** (superclass) untuk merujuk pada objek **subtype** (subclass).

Contoh: Kendaraan p = new Pesawat(); adalah valid jika Pesawat adalah subclass dari Kendaraan.

Dua Wajah Polymorphism

Ini adalah dua cara utama polimorfisme diimplementasikan di Java:

Overloading

(Compile-time Polymorphism / Static Binding)

- Terjadi dalam **satu kelas yang sama**
- Method memiliki **nama sama** tetapi **parameter berbeda**
- Contoh: hitung(int a) dan hitung(double a, double b)

Overriding

(Run-time Polymorphism / Dynamic Binding)

- Terjadi antara **superclass dan subclass**
- Method di subclass memiliki **nama, parameter, dan tipe kembalian yang sama persis**
- Memungkinkan subclass memberikan implementasi spesifik

BAGIAN 4

**Uji Kemampuan & Contoh
Soal Coding**

Uji Kemampuan 1: Konsep Dasar & Constructor

Studi Kasus: Kelas Buku

1 Buat Kelas Buku

Tambahkan atribut private: judul (String), pengarang (String), dan harga (double)

2 Buat Dua Constructor

Constructor pertama tanpa parameter dengan nilai default. Constructor kedua dengan tiga parameter untuk inisialisasi semua atribut

3 Method cetakInfo()

Buat method public void cetakInfo() yang menampilkan semua detail buku ke konsol

4 Testing di Main Method

Buat objek buku1 dengan constructor tanpa parameter dan buku2 dengan data "Laskar Pelangi", "Pelangi", "Andrea Hirata", 75000

Uji Kemampuan 2: Encapsulation & Validasi

Studi Kasus: Melengkapi Kelas Buku

01

Pastikan Encapsulation

Semua atribut menggunakan access modifier private

02

Buat Getter dan Setter

Method Getter dan Setter public untuk setiap atribut (judul, pengarang, harga)

03

Tambahkan Validasi

Di setHarga(), jika hargaBaru negatif, jangan ubah nilai dan nilai dan cetak pesan error "Harga tidak boleh negatif!"
negatif!"

04

Testing Validasi

Uji dengan nilai positif dan negatif untuk memastikan memastikan validasi berfungsi

Uji Kemampuan 3: Inheritance (Pewarisan)

Buat Superclass Pegawai

Atribut protected String nama dan double gajiPokok.
gajiPokok. Constructor yang menerima nama dan
dan gajiPokok. Method public void cetakInfo()

1

Constructor Manager

Menerima nama, gajiPokok, dan tunjangan. Gunakan
Gunakan keyword super() untuk memanggil
constructor Pegawai

2

Buat Subclass Manager

Extends dari Pegawai. Tambahkan atribut private
private double tunjangan

3

Testing

Buat objek Manager dan panggil method cetakInfo()

4

Override Method

Override method cetakInfo() di Manager untuk
menampilkan nama, gaji pokok, dan tunjangan

5

Uji Kemampuan 4: Polymorphism & Array of Objects

Studi Kasus: Koleksi BangunDatar

Buat Superclass BangunDatar

Method public double hitungLuas() yang mengembalikan 0 dan public void info() yang mencetak "Ini adalah bangun datar"

Buat Tiga Subclass

Persegi, Lingkaran, dan Segitiga yang extends BangunDatar. Tambahkan atribut yang relevan untuk masing-masing

Override Methods

Override method hitungLuas() dan info() di setiap subclass untuk implementasi yang sesuai dengan bentuknya

Array Polymorphism

Buat array BangunDatar[] dan isi dengan objek Persegi(5), Lingkaran(7), dan Segitiga(6, 8)

Testing Polymorphism

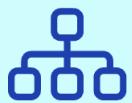
Gunakan perulangan untuk memanggil method info() dari setiap objek dan amati output yang berbeda

Best Practices dalam PBO



Gunakan Encapsulation

Selalu buat atribut private dan sediakan getter/setter dengan validasi yang tepat untuk melindungi data



Desain Inheritance dengan Hati-hati

Pastikan hubungan "is-a" benar-benar valid. Jangan gunakan inheritance hanya untuk code reuse



Manfaatkan Polymorphism

Gunakan polymorphism untuk membuat kode yang fleksibel dan mudah diperluas



Penamaan yang Konsisten

Gunakan konvensi penamaan Java: class dengan PascalCase, method dan variabel dengan camelCase

Kesalahan Umum dalam PBO

Melanggar Encapsulation

Membuat atribut public atau tidak menyediakan validasi dalam dalam setter

Inheritance yang Salah

Menggunakan inheritance untuk untuk hubungan "has-a" instead instead of "is-a"

Lupa super() dalam Constructor

Tidak memanggil constructor superclass di baris pertama constructor subclass

Overriding vs Overloading

Bingung antara overriding (inheritance) dan overloading (same class)

Tidak Menggunakan @Override

Tidak menggunakan annotation @Override saat melakukan method overriding

Rangkuman & Tanya Jawab

Fondasi PBO

Class & Objek adalah fondasi, didukung oleh **Atribut & Method**
Method

Flexibility

Polymorphism memberikan fleksibilitas melalui overloading dan overriding

Code Reuse

Inheritance menciptakan hierarki dan mengurangi duplikasi kode



Tools Java

Dasar Java seperti **Tipe Data**, **Operator**, dan **Struktur Kontrol** adalah alatnya

Data Collection

Array memungkinkan penyimpanan penyimpanan koleksi data sejenis sejenis

Object Builder

Constructor membangun dan menginisialisasi objek

Data Protection

Encapsulation melindungi data dengan private dan setter/getter

Ada Pertanyaan?

PBO mengubah cara kita memandang program dari urutan instruksi menjadi interaksi antar objek. Dengan memahami konsep-konsep fundamental ini, Anda telah memiliki dasar yang kuat untuk pengembangan perangkat lunak yang lebih terstruktur dan maintainable.

Praktik Terus Menerus

Implementasikan konsep-konsep yang telah dipelajari dalam project nyata

Eksplorasi Lebih Lanjut

Pelajari konsep advanced seperti Abstract Class, Interface, dan Design Patterns

Bangun Portfolio

Buat aplikasi sederhana yang menerapkan semua prinsip PBO yang telah dipelajari